

# CCTBX tools for derivative-free optimization

Haiguang Liu<sup>1,2</sup>, Billy Poon<sup>1</sup>, Gang Chen<sup>1</sup>, Ralf Grosse-Kunstleve<sup>3</sup> & Peter Zwart<sup>1\*</sup>

*1. Berkeley Center for Structural Biology, Lawrence Berkeley National Laboratories, Physical Biosciences Division*

*2. Currently at Arizona State University, Department of Physics*

*3. Computational Crystallographic Initiative, Lawrence Berkeley National Laboratories, Physical Biosciences Division*

Email: [PHZwart@lbl.gov](mailto:PHZwart@lbl.gov)

## Introduction

Many applications developed in crystallography, small angle scattering and general scattering sciences require the determination of model parameters given experimental data (Afonine *et al.*, 2005; Liu *et al.*, 2012). In crystallography, gradient-based optimizers such as L-BFGS (Liu & Nocedal, 1989) have proven to be quite successful for handling challenges associated with large-scale optimization problems such as structure refinement. Whereas the L-BFGS and other Newton-based optimizers available in CCTBX have proven their utility (Bourhis *et al.*, 2007), the drawback of these methods is the requirement that the starting solution is close to the global minimum of the target function, and that the path from the starting location towards the global minimum does not lead through local minima in which gradient-based methods may get trapped. From a software development point of view, an additional drawback is the need to implement numerically stable and well-tested first (and second) derivatives. Deriving, implementing and testing gradients and second derivatives can often be time consuming relative to the problem one aims to solve, especially when the target functions are involved or when one has to deal with domain restrictions for the parameters. In cases where the optimization of the parameters is not a (perceived) time-limiting step in the total, the use of derivative-free optimizers can result in straightforward and robust code that gets the job done. In this article, we highlight the derivative-free optimization methods available in the CCTBX and provide a brief comparison of their performance.

## Available Methods

In the following sections, five different derivative-free optimization methods are outlined. For completeness, we provide a brief overview of the L-BFGS gradient-based optimizer as well. The code snippets are taken from

`scitbx/examples/minimizer_examples.py`, as distributed with CCTBX from January 19, 2012 onwards.

### Gradient based methods

Available gradient-based minimisation methods in CCTBX are

1. L-BFGS (`scitbx.lbfsgs`)
2. Damped Newton  
(`scitbx.minimizers.damped_newton`)
3. Newton with More-Thuente line search  
(`scitbx.newton_more_thuente_1994`)

The L-BFGS minimizer uses first derivatives only, but builds up an estimate of the inverse of the Hessian based on the BFGS formula (Nocedal & Wright, 2006) using information from previous steps. For large-scale optimization problems for which first derivatives are available, this minimizer is typically a good choice. The Newton-type minimizers (with and without line search) require second derivative information. These minimizers are well suited for problems in which the variables have a large spectrum of scales, or problems with strongly correlated parameters. By design, these gradient-based minimizers perform a local search only.

### Simplex Search

A standard simplex algorithm is available for local optimization purposes. The simplex optimization algorithm was first proposed by Nelder and Mead (Nelder & Mead, 1965). A modified version is adopted in the current CCTBX implementation. A pure-Python implementation of this algorithm is available in `scitbx.simplex`. A prototype use of this optimizer is shown in Scheme 1.

### Direct-Search Simulated Annealing (DSSA)

Combining a simplex search with simulated annealing results in a versatile and robust optimizer that is able to tackle a wide variety of optimization problems (Hedar & Fukushima, 2002). Due to the introduction of the annealing

```

class test_simplex():
    def __init__(self, name):
        self.n = 2; self.x = flex.double(self.n, 2)
        self.target = Function(name)(dim).eval
        self.starting_simplex=[]
        for ii in range(self.n+1):
            self.starting_simplex.append(
                (flex.random_double(self.n)/2-1)*0.1+ self.x)
        self.optimizer = simplex.simplex_opt( dimension=self.n,
                                              matrix = self.starting_simplex,evaluator=self,tolerance=1e-10)
        self.x = self.optimizer.get_solution()

```

**Schema 1:** Simplex method

```

class test_dssa():
    def __init__(self,name):
        self.n = 2
        self.x = flex.double(self.n, 2)
        self.target = Function(name)(dim).eval
        self.starting_simplex=[]
        for ii in range(self.n+1):
            self.starting_simplex.append(
                0.01*(flex.random_double(self.n)/2-1) + self.x)
        self.optimizer = direct_search_simulated_annealing.dssa(
            dimension=self.n,
            matrix = self.starting_simplex,
            evaluator = self,
            tolerance=1e-8,
            further_opt=True,n_candidate=None,
            coolfactor=0.5, simplex_scale=1
        )
        self.x = self.optimizer.get_solution()

```

**Schema 2:** DSSA method

procedure, the algorithm is less likely to get stuck in local minima. Furthermore, the rate of convergence is enhanced by the use of a local search. The interface to the DSSA algorithm is similar to that of the simplex method and is found in `scitbx.direct_search_simulated_annealing` (See Schema 2). Although the algorithm is mostly self-steering, important parameters are the cooling factor and the choice of the initial simplex size.

### Cross Entropy (CE)

The cross-entropy method for optimization (Rubenstein, 1997) is a Monte Carlo based optimization method. In the CE method, a prior probability distribution of the refinable parameters is defined from which candidate solutions are drawn. Each candidate solution is subsequently used to compute calculated data. Using only a small fraction of candidate solutions (the elite set) that provide a best fit to the data, the prior distribution is modified using the elite candidates. The interface to this optimizer requires the definition of a prior distribution by providing estimates of the mean and standard deviation of the proposed parameters.

The updates to the sampling distribution are performed on the mean and standard deviations only.

```

class test_cross_entropy():
    def __init__(self, name):
        self.n = 2
        self.x = flex.double(self.n, 2)
        self.target = Function(name)(dim).eval
        self.means = flex.double( self.n, 4.0 )
        self.sigmas = flex.double( self.n, 2.0 )
        self.optimizer = cross_entropy.cross_entropy_optimizer(self,
                                                               mean=self.means,
                                                               sigma=self.sigmas,
                                                               alpha=0.75,
                                                               beta=0.75,
                                                               q=8.5,
                                                               elite_size=10,
                                                               sample_size=100)

```

**Schema 3:** Cross Entropy method

```

class test_cma_es():
    def __init__(self, name):
        self.m = flex.double( [2,2] )
        self.s = flex.double( [1,1] )
        self.name = name
        self.minimizer = cma_es_interface.cma_es_driver(
            2, self.m, self.s, self.my_function )
        self.best_solution = self.minimizer.x_final

    def my_function(self, vector):
        tmp = Function(self.name)(dim)
        return tmp.eval(list(vector))

```

**Schema 4:** Covariance Matrix Adaption – Evolution Strategy

Estimates of the correlation between parameters are not taken into account. Important parameters to experiment with are the elite and sample sizes, as well as the learning rates, alpha and beta. The learning rates, which range between 0 and 1, affect the steps sizes when updating the means (alpha) and standard deviations (beta).

#### Covariance Matrix Adaptation - Evolution Strategy (CMA-ES)

CMA-ES is another Monte Carlo based optimization method (Hansen, 2003). In essence, it is very similar to the CE method outlined above, but has two major advantages. First of all, contrary to the CE method, CMA-ES actively explores correlations between parameters. A second advantage is that it provides adaptive step-size control in order to prevent preliminary convergence. The CMA-ES method starts with a

trial Gaussian distribution assuming no dependence between input parameters. In each optimization cycle, a set of candidate solutions is drawn from this sampling distribution. The candidate solutions are sorted according to their correspondence to the data. At this point, a weighted mean is computed from these candidates, with weights proportional to their ranks only. This updated mean is an improved estimate over the previous estimated mean. In a similar manner, updates to the estimated covariance matrix are performed. Integral to the distribution updates is the determination of the step size. The adaptive step-size control protocol prevents premature convergence of standard deviations to very small values. This prevents the algorithm from premature convergence as is sometimes seen in the CE method. An example

```

class test_differential_evolution():
    def __init__(self, name):
        self.n = 2
        self.x = None #flex.double(self.n, 2)
        self.target = Function(name)(dim).eval
        self.domain = [(start[0]-2,start[0]+2), (start[1]-2, start[1]+2)]
        self.optimizer=differential_evolution.differential_evolution_optimizer(
            self,population_size=20,cr=0.9,n_cross=2)

```

### Schema 5: Differential Evolution

interface is found below:

The algorithm is virtually free of parameters that needs to be tuned, in contrast to the DSSA or CE algorithms.

### Differential Evolution (DE)

Differential evolution is yet another population-based, Monte Carlo type optimization algorithm (Storn & Price, 1997). In the DE algorithm, new candidate solutions are constructed around an existing solution by perturbations generated from the difference between two other candidate solutions. Over time, an initial population with large spread contracts around the (hopefully) global minimum. The interface is similar to most other optimizers discussed in this article:

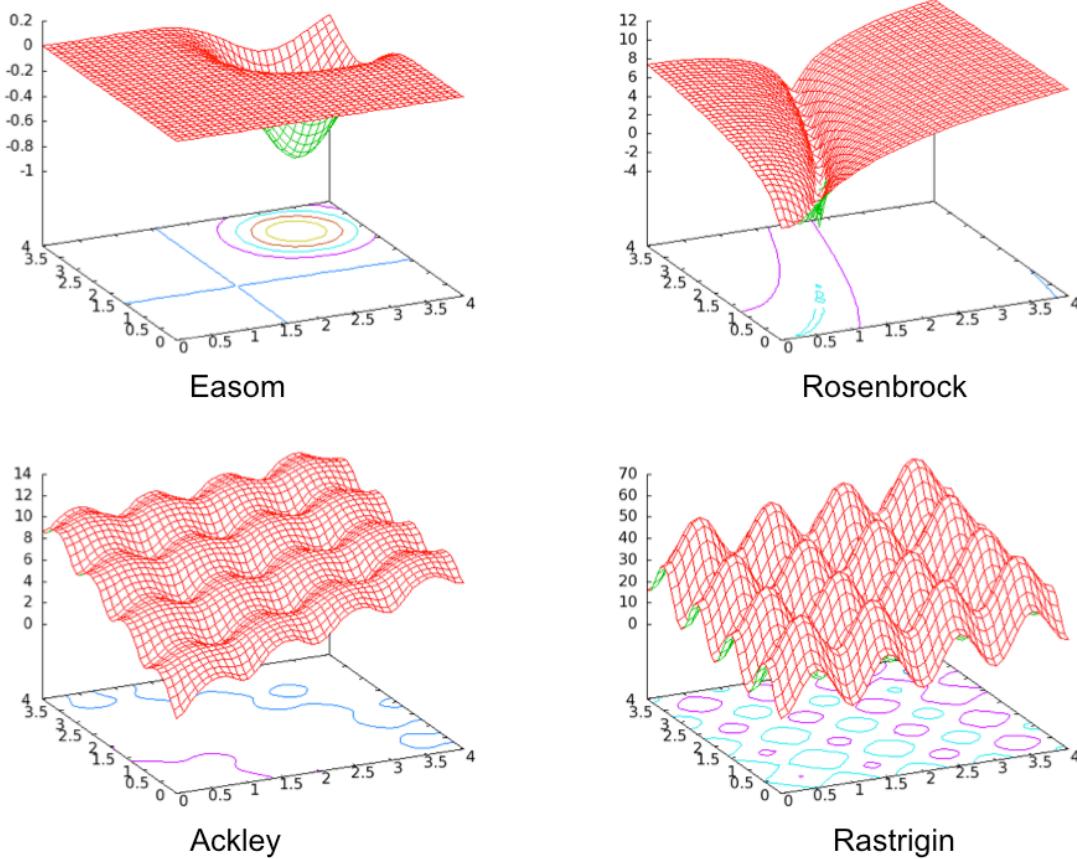
As in the case for CMA-ES, DE can be used virtually as a black box. The only important parameter is the definition of the domain in which the solution is likely to be found. This definition does not necessarily need to contain the minimum.

## Results

To test the performance of these algorithms, four test functions (Easom, Rosenbrock, Ackley, Rastrigin) were optimized with the derivative free-optimization methods. For comparison, L-BFGS with finite difference derivatives (FD-L-BFGS) trials were performed as well. Apart from the Rosenbrock function, all test functions have multiple local minima. The test functions are depicted in Figure 1.

**Table 1:** Average and standard deviation for the number of function calls before convergence for 1000 trials. Italicised numbers indicate that the global minimum was not found. All minimizers were run with typical settings.

Optimizer / Function	Easom	Rosenbrock	Ackley	Rastrigin
<b>FD-L-BFGS</b>	36	120	100	20
<b>Simplex</b>	$195 \pm 20$	$403 \pm 84$	$124 \pm 96$	$126 \pm 16$
<b>CE</b>	$17087 \pm 4605$	$13633 \pm 9722$	$18745 \pm 4894$	$14115 \pm 4894$
<b>DSSA</b>	$379 \pm 96$	$884 \pm 37$	$561 \pm 128$	$476 \pm 21$
<b>DE</b>	$792 \pm 80$	$1525 \pm 150$	$2596 \pm 181$	$2140 \pm 349$
<b>CMA-ES</b>	$551 \pm 128$	$820 \pm 118$	$976 \pm 69$	$745 \pm 100$



**Figure 1:** Test functions for optimization routines.

evaluations, while CMA-ES (with a population size of 20) converges to the correct answer within 27000 function evaluations. Since the Rosenbrock function is convex, the FD-L-BFGS algorithm outperforms CMA-ES by using only 1100 function evaluations.

### Discussion and Conclusions

Optimization, very often involving non-quadratic target functions, plays an important role in many scientific procedures. When confronted with an optimization problem, typically two choices emerge. Either one has to estimate a starting solution, sufficiently close to the global minimum, that is within the reach of gradient-based local

minimization methods. Or alternatively, one has to treat the optimization problem as a (semi-)global problem and use non-local optimizers to find the solution. The `scitbx` module of CCTBX includes functionality for both choices as illustrated in this article. The derivative-free optimizers are typically outperformed by the L-BFGS minimizer on convex functions. For (semi-)global optimization problems, both DE and CMA-ES seem to be suitable choices as they require little tuning and both have good convergence properties. The availability of these optimizers in `scitbx` removes important bottlenecks in scientific software development.

**Table 2:** Success rate of DE and CMA-ES for 2D-Rastrigin global optimization problem.

Population size	DE	CMA-ES
5	0.01	0.24
10	0.27	0.45
15	0.67	0.58
20	0.88	0.71
40	1.00	0.91

## References

- Afonine, P.V., Grosse-Kunstleve, R.W. & Adams, P.D. (2005). *CCP4 Newsletter on Protein Crystallography*, **42** (8).
- Bourhis, L.J., Grosse-Kunstleve, R.W. & Adams, P.D. (2007). *Newsletter of the IUCr Commission on Crystallographic Computing*, **8**, 74-80.
- Hansen, N., Müller, S.D. & Koumoutsakos, P. (2003). *Evolutionary Computation*, **11**, 1-18.
- Hedar, A. & Fukushima, M. (2002). *Optimization Methods and Software*, **17**, 891-912.
- Liu, D.C. & Nocedal, J. (1989). *Mathematical Programming B* **45** (3), 503-528.
- Liu, H., Morris, R. J., Hexemer, A., Grandison, S. & Zwart, P.H. (2012). *Acta Crystallographica A* (in press).
- Nelder, J.A. & Mead, R. (1965). *The Computer Journal*, **7**, 308-313.
- Nocedal, J. & Wright, S.J. (2006). *Numerical Optimization, 2nd Edition*, Springer Verlag.
- Rubinstein, R.Y. (1997). *European Journal of Operations Research*, **99**, 89-112.
- Storn, R. & Price, K. (1997). *Journal of Global Optimization*, **11**, 341-359.